Lab 4

Reg. No: 19BCE1209

Name: Gautam Sanjay Wadhwani

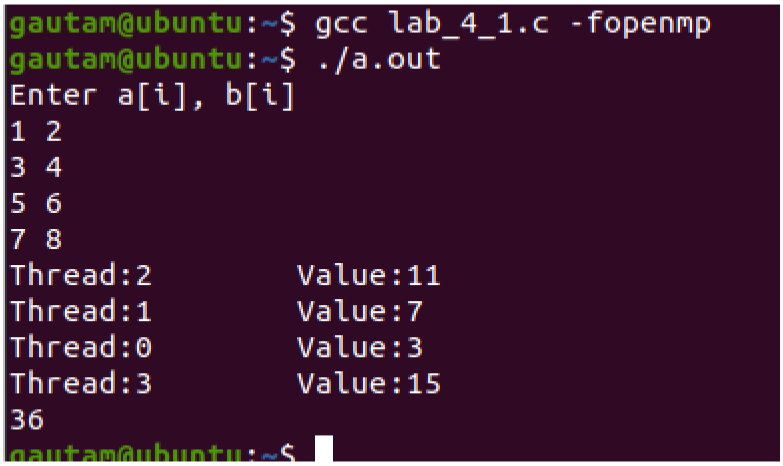
Course: CSE4001 Parallel and Distributed Computing

Q1. Sample for Reduction

Code:

```
#include <stdio.h>
#include <omp.h>
int main ()
{
int a[4],b[4],c[4];
int i, sum = 0;
printf("Enter a[i], b[i]\n");
for(i = 0; i < 4; i++) {
scanf("%d %d", &a[i], &b[i]);
}
omp_set_num_threads(4);
#pragma omp parallel for reduction(+:sum)
for (i=0; i < 4; i++)
{
c[i] = a[i] + b[i];
sum += c[i];
printf("Thread:%d\tValue:%d\n",omp_get_thread_num(),c[i]);
}
printf("%d\n", sum);
return 0;
}
```

Output:



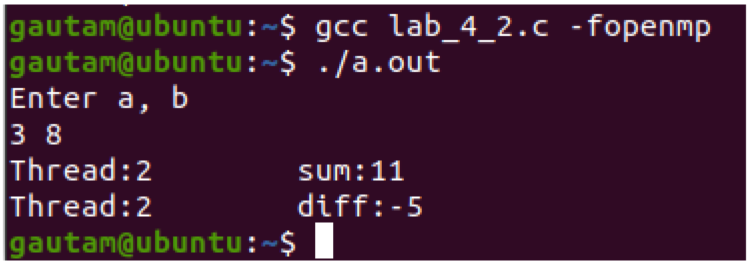
```
gautam@ubuntu:~$ gcc lab_4_1.c -fopenmp
gautam@ubuntu:~$ ./a.out
Enter a[i], b[i]
1 2
3 4
5 6
7 8
Thread:2        Value:11
Thread:1        Value:7
Thread:0        Value:3
Thread:3        Value:15
36
gautam@ubuntu:~$
```

Q2. Parallel : c = a+b; c= a-b

Code:

```
#include <stdio.h>

#include <omp.h>

int main ()

{

int a,b,c;

printf("Enter a, b\n");

scanf("%d %d", &a, &b);

omp_set_num_threads(4);

#pragma omp parallel sections private(c)

{

#pragma omp section

{

int sum = a+b;

printf("Thread:%d\tsum:%d\n",omp_get_thread_num(),sum);

}

#pragma omp section

{

int diff = a-b;
printf("Thread:%d\tdiff:%d\n",omp_get_thread_num(),diff);

}

}

return 0;

}
```

Output:



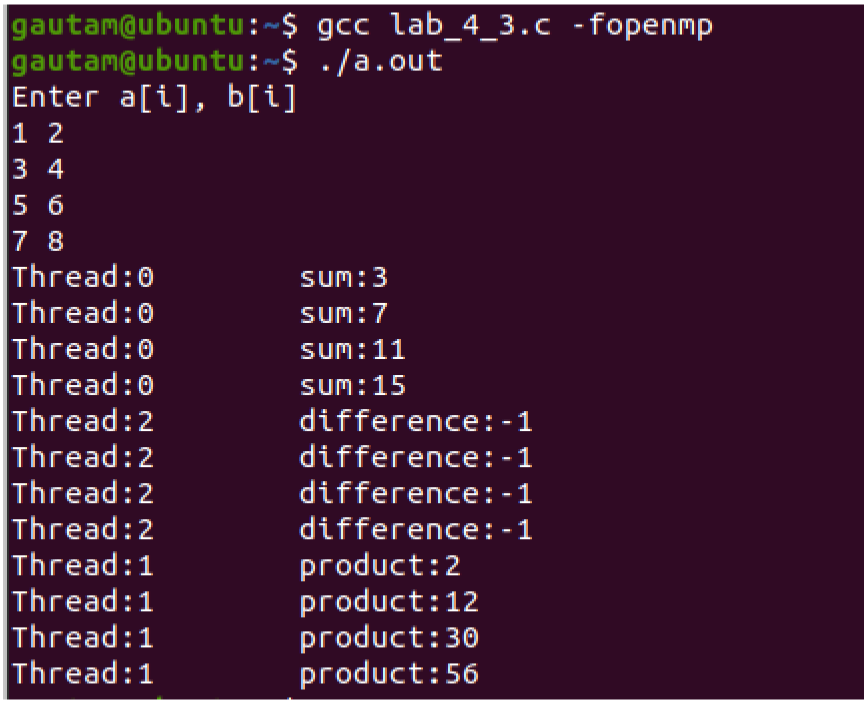
```
gautam@ubuntu:~$ gcc lab_4_2.c -fopenmp
gautam@ubuntu:~$ ./a.out
Enter a, b
3 8
Thread:2        sum:11
Thread:2        diff:-5
gautam@ubuntu:~$
```

Q3. Parallel : c[i] = a [i]+b[i] ; c [i] =a[i]*b[i] ; c[i]-b[i]

Code:

```
#include <stdio.h>

#include <omp.h>

int main ()

{

int a[4],b[4],c[4];

printf("Enter a[i], b[i]\n");

for(int i = 0; i < 4; i++)

scanf("%d %d", &a[i], &b[i]);

omp_set_num_threads(4);

#pragma omp parallel sections

{

#pragma omp section

{

for(int i = 0; i < 4; i++)

{

c[i] = a[i]+b[i];
printf("Thread:%d\tsum:%d\n",omp_get_thread_num(),c[i]);
}
}
#pragma omp section
{
for(int i = 0; i < 4; i++)
{
c[i] = a[i]*b[i];
printf("Thread:%d\tproduct:%d\n",omp_get_thread_num(),c[i]);
}
}
#pragma omp section
{
for(int i = 0; i < 4; i++)
{
c[i] = a[i]-b[i];
printf("Thread:%d\tdifference:%d\n",omp_get_thread_num(),c[i]);
}
}
}
return 0;
}
```

Output:



```
gautam@ubuntu:~$ gcc lab_4_3.c -fopenmp
gautam@ubuntu:~$ ./a.out
Enter a[i], b[i]
1 2
3 4
5 6
7 8
Thread:0        sum:3
Thread:0        sum:7
Thread:0        sum:11
Thread:0        sum:15
Thread:2        difference:-1
Thread:2        difference:-1
Thread:2        difference:-1
Thread:2        difference:-1
Thread:1        product:2
Thread:1        product:12
Thread:1        product:30
Thread:1        product:56
```

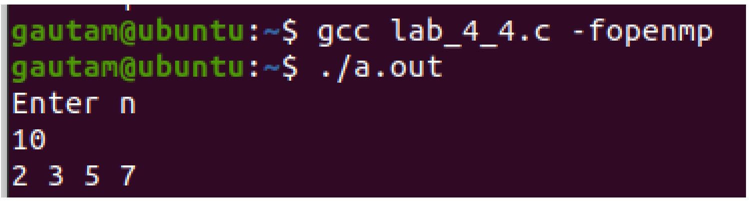
Q4. Implement listing of prime numbers < N

Code:

```
#include <stdio.h>

#include <omp.h>

int main ()

{

int n;

printf("Enter n\n");

scanf("%d", &n);

int prime[n];

for(int i = 0; i < n+1; i++) prime[i] = 1;

for(int i = 4; i < n+1; i += 2) prime[i] = 0;

omp_set_num_threads(4);

#pragma omp parallel for shared(prime)

for(int i = 3; i < n+1; i+= 2)

{

if(prime[i] == 1)

{
```

```
for(int j = i*i; j < n+1; j += i)
{
prime[j] = 0;
}
}
}
for(int i = 2; i < n+1; i++) if(prime[i]) printf("%d ", i);
printf("\n");
return 0;
}
```

Output:



```
gautam@ubuntu:~$ gcc lab_4_4.c -fopenmp
gautam@ubuntu:~$ ./a.out
Enter n
10
2 3 5 7
```

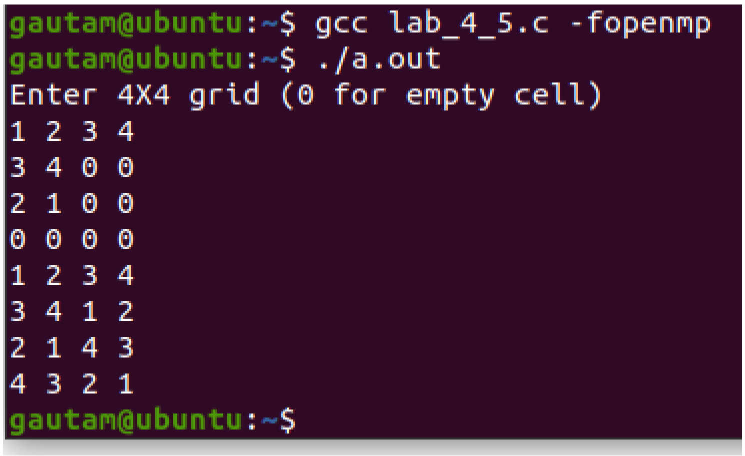
Q5. Implement "Sudoku solving algorithm (2*2) (16 cells)

Code:

```
#include <stdio.h>
#include <omp.h>
void display(int grid[4][4]) {
   for(int i = 0; i < 4; i++) {
      for(int j = 0; j < 4; j++) {
         printf("%d ", grid[i][j]);
      }
      printf("\n");
   }
   printf("\n");
}
int check(int grid[4][4], int i, int j) {
   for(int k = 0; k < 4; k++) {
       if(grid[i][j] == grid[i][k] || grid[i][j] == grid[k][j]) return 0;
   }
   return 1;
}
int solve(int grid[4][4], int x) {
   if(x == 16) {
       display(grid);
       return 1;
   }
   int i = x/4, j = x%4, solved = 0;
   if(grid[i][j] == 0) {
       #pragma omp parallel for shared(grid, x, i, j, solved)
       for(int k = 1; k < 10; k++) {
           grid[i][j] = k;
           if(check(grid, i, j) && solve(grid, x+1)) {
               solved = 1;
           }
           grid[i][j] = 0;
       }
   }
   return solved;
}
int main ()
{
int n = 4;
int grid[4][4];
printf("Enter 4X4 grid (0 for empty cell)\n");
for(int i = 0; i < 4; i++)
{
for(int j = 0; j < 4; j++)
{
```

```
scanf("%d", &grid[i][j]);

}

}

solve(grid, 0);

return 0;

}
```

Output:



```
gautam@ubuntu:~$ gcc lab_4_5.c -fopenmp
gautam@ubuntu:~$ ./a.out
Enter 4X4 grid (0 for empty cell)
1 2 3 4
3 4 0 0
2 1 0 0
0 0 0 0
1 2 3 4
3 4 1 2
2 1 4 3
4 3 2 1
gautam@ubuntu:~$
```